

# **K, the Fourth Order Coefficient Tensor Used in ALE3D's Quadratic Generalized von Mises Yield Function, in Five Easy Steps**

*M.J. Busche*

**August 11, 2000**

***U.S. Department of Energy***

Lawrence  
Livermore  
National  
Laboratory

## **DISCLAIMER**

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This work was performed under the auspices of the U. S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48.

This report has been reproduced  
directly from the best available copy.

Available to DOE and DOE contractors from the  
Office of Scientific and Technical Information  
P.O. Box 62, Oak Ridge, TN 37831  
Prices available from (423) 576-8401  
<http://apollo.osti.gov/bridge/>

Available to the public from the  
National Technical Information Service  
U.S. Department of Commerce  
5285 Port Royal Rd.,  
Springfield, VA 22161  
<http://www.ntis.gov/>

OR

Lawrence Livermore National Laboratory  
Technical Information Department's Digital Library  
<http://www.llnl.gov/tid/Library.html>

# **K, the Fourth Order Coefficient Tensor Used in ALE3D's Quadratic Generalized Von Mises Yield Function, in Five Easy Steps**

Matt Busche  
New Technologies Engineering Division  
Lawrence Livermore National Laboratory  
Livermore, CA 94550, USA

## **ABSTRACT**

This document describes the software developed for use in calculating  $K$ , the 4<sup>th</sup> order parameter tensor used in ALE3D's anisotropic plasticity model. The multi-scale modeling method developed for this calculation begins with orientation imaging microscopy (OIM) data. The program OIMA3D characterizes the sizes and crystal orientation of the grains found in this data and then determines element orientations for a representative 3D mesh. A shell script, MAKEJOBS, then creates the necessary files to run six ALE3D simulations using this mesh. The results of these simulations are then read by SVD\_K, a Matlab script, and  $K$  is calculated from this information.

<b>Step</b>		<b>page</b>
<b>1</b>	Trim and clean OIM data as needed.	3
<b>2</b>	Run <b>OIMA3D</b> to assign crystal orientations to mesh.	4
<b>3</b>	Run <b>MAKEJOBS</b> to prepare ALE3D jobs.	10
<b>4</b>	Submit jobs and wait.	
<b>5</b>	Run <b>SVD_K</b> in Matlab to calculate K.	11

## Data Cleanup

Figure 1(a) shows a grain boundary map of a typical OIM scan as generated in TexSEM Laboratories' OIM Analysis for Windows. The file contains many erroneous data points that need to be removed. The bottom of the file contains meaningless data which was scanned beyond the edge of the specimen, and many erroneous points caused by surface scratches are visible. To improve the scan, first trim the bad data from the edges. In the case shown the bad data at the bottom as well as a small margin on the left side was removed. This is accomplished by right clicking on the map and selecting "export scan data". A dialogue will request a filename, and then a crosshair will appear. Use this to drag a box over the portion of the scan you wish to retain. After obtaining the newly trimmed data file, the next step is to clean up the erroneous points. Open the trimmed file and select "Data Cleanup" under the Tools menu. A set of options that works well is "Grain Dilation" with a grain tolerance angle of  $15^\circ$  and a minimum grain size of 10. Figure 1(b) shows the results of trimming and cleaning the data.

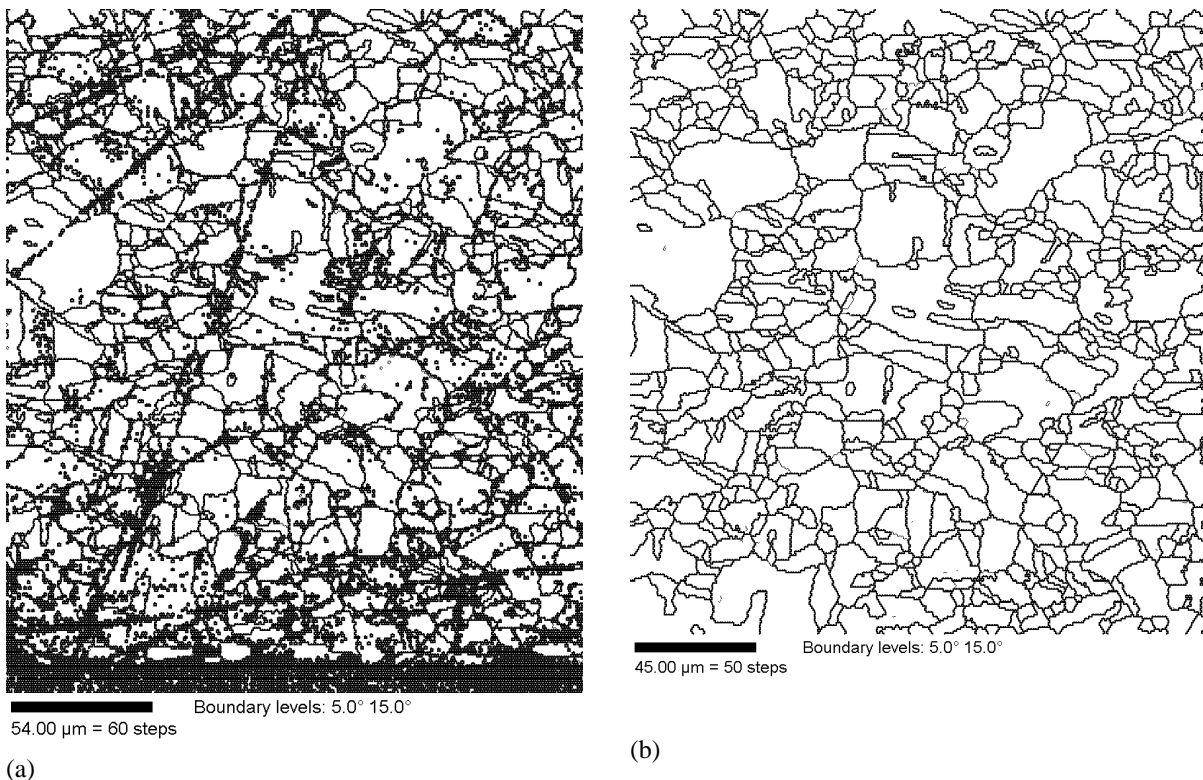


Figure 1. Comparison of (a) as scanned and (b) cleaned OIM data.

## **OIMA3D – mapping Orientation Imaging Microscopy data to Ale3D element meshes.**

### **Overview**

OIMA3D generates a set of element orientations based on OIM scan data for use in ALE3D. The OIM data can come from two possible sources, a grain data file or directly from the scan data. Based on the grain properties of the input, the program generates grains in the element mesh which retain the orientations and relative size of those found in the input and then arranges them using a simulated annealing algorithm. An ordered list of the element orientations is output which is read by ALE3D's crystal orientation method 4.

### **Usage**

With a grain data file generated by TexSem OIM Analysis (type 2) or by OIMA3D:

```
OIMA3D -g <grain data filename> <output filename>
```

Directly from OIM scan data (.ang file):

```
OIMA3D -a <ang filename> <output filename>
```

### **Input**

The program was originally written to work directly from a grain data file generated in TexSEM Laboratories' OIM Analysis for Windows. The format of the grain data file is shown in Figure 2. Note that only the Euler angles and radius are used by OIMA3D.

Figure 2. Format of grain data file.

phi1	PHI	phi2	x	y	Image quality	Confidence index	area	radius
------	-----	------	---	---	------------------	---------------------	------	--------

However, it was found that the algorithm in TexSEM OIM Analysis used to create the grain data files grossly distorted the texture of the data. To remedy this, an algorithm was implemented in OIMA3D which reads an OIM scan file and creates a grain data file that retains the texture of the data. This algorithm assigns every data point in the file to a grain based on two criteria, orientation and proximity.

As a point is evaluated, its orientation is compared to each existing grain. If phi1 and phi2 of the point are both within 0.15 radians of phi1 and phi2 of an existing grain, and the point is adjacent to any existing point in that grain, then the new point is added to the grain. If a data point has a unique orientation or is not adjacent to any of the points in

a similarly orientated grain, then a new grain is created. Note that “adjacent” is implemented as meaning within the distance of three points to allow for roughness in the data. The orientation of a grain is determined as a running average of the points added to it. This algorithm accounts for every point in the data file, and therefore retains the texture of the data file very well. As a final step, the grain data file is created. The radius field is the square root of the number of points in each grain. This provides a length scale representative of the area fraction of the grains.

The option for input of a grain data file is retained to allow the use of a grain data file generated in a previous run of OIMA3D or a TexSEM grain data file if desired.

## Operation

Operation begins with user input of five parameters: mesh dimensions (number of elements) in x, y, and z, the desired average number of elements per grain, and the grain size exponent. With these parameters acquired, OIMA3D begins by generating the grain data structures. This data structure contains the Euler angles of the grain, the global position of the grain within the mesh, and an array of coordinates relative to the global position that define the discrete points that make up the grain.

The grain data structures are created based on the data in the grain data file that was either created earlier or specified on the command line. The routine reads through the file sequentially, looping back to the beginning if necessary. Three things are added to each grain data structure at this point: the Euler angles, an initial position, and the array of element coordinates. The Euler angles are read from the grain data file, and the initial position is a random point in the mesh. To create the array of elements, its size must first be determined.

The number of elements in a grain is proportional to the length scale found in the radius field of the grain data file raised to the grain size exponent input earlier. However, the length scales must be normalized so that the desired average number of elements per grain is achieved. The number of elements in a grain is defined by

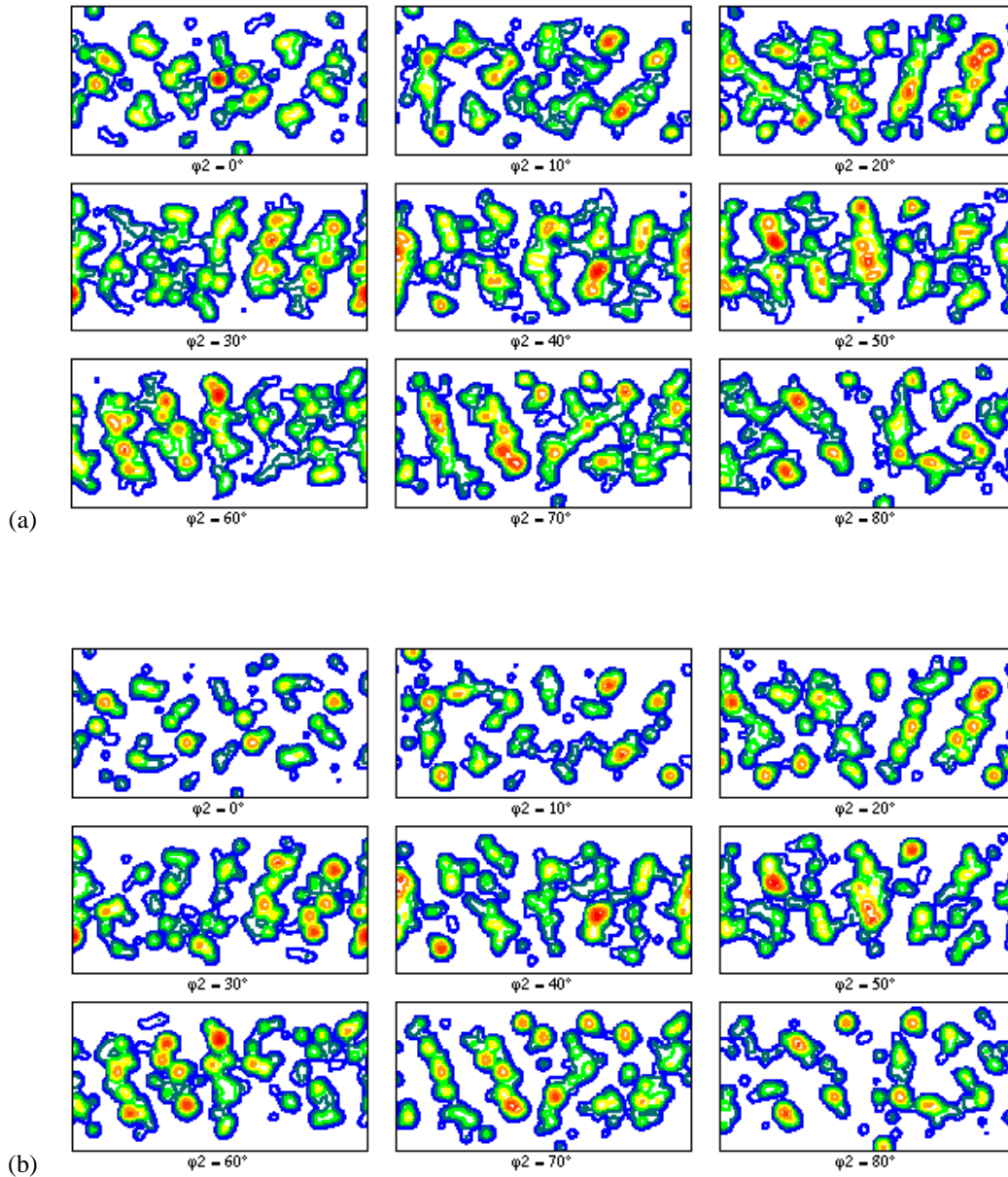
$$E_g = \left( \frac{l}{\bar{l}} \right)^m \bar{E} \quad (1)$$

where  $\bar{E}$  is the average number of elements per grain and m is the grain size exponent, both input earlier, and with the average length scale

$$\bar{l} = \sqrt[m]{\frac{1}{n} \sum l^2} \quad (2)$$

Note that grains which scale to less than one element are not used in computing the average length scale, which makes it necessary to solve (2) iteratively. The number of grains created in the mesh is controlled so that the sum of the elements in the grains is roughly 30% larger than the number of elements in the mesh

The grain size exponent is a user specified value because it's not entirely clear what the appropriate value is. Using a value of two makes the volume fractions in the mesh equal to the area fractions in the OIM data. However, using a value of three seems to be more physically appropriate. Figure 3 compares contour ODFs for the actual OIM data versus meshes using grain exponents of two and three. The value of the grain size exponent can be any real number greater than one.



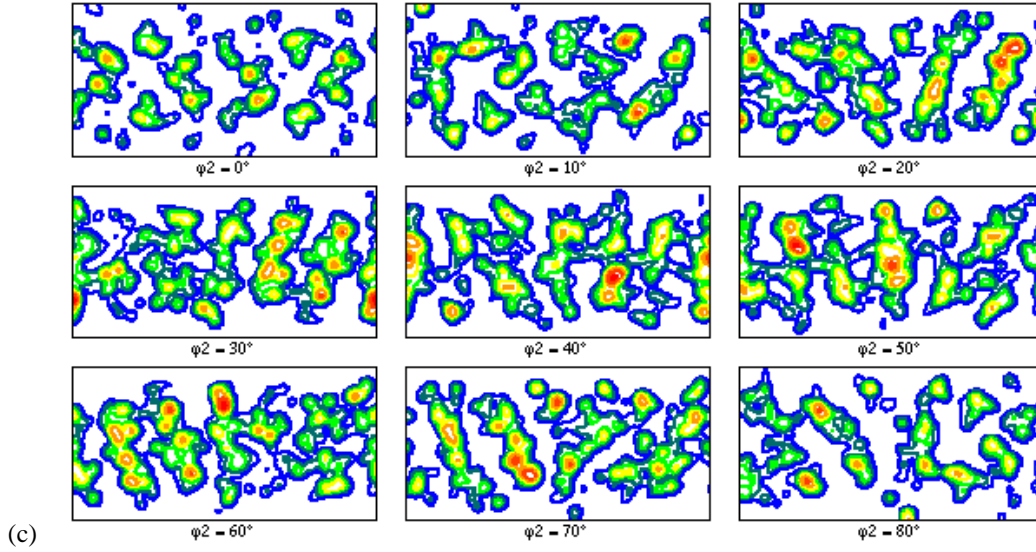


Figure 3. Contour ODFs of (a) OIM scan data (b) element mesh with  $m = 3$   
(c) element mesh with  $m = 2$ .

The final step in creating the grain data structures is to assign the specified number of element coordinates to each grain. The first eight elements are assigned in the progression shown in Figure 4(a). The next twelve elements are in the set shown in Figure 4(b), but are assigned in a random order. Any additional elements beyond these first twenty are assigned to a random contiguous point. This method for initializing the grains is not ideal. It was desired that the grains show some randomness in their shape, but the grains should be relatively equiaxed. The major disadvantage of the method implemented is that the grains “grow” in only three directions.

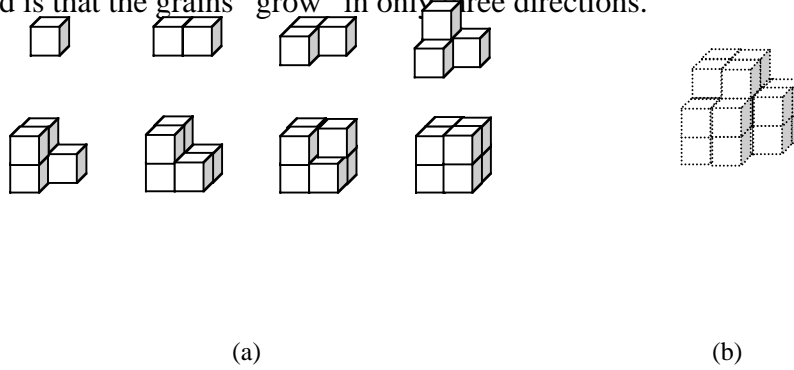


Figure 4. Progression of grain shape.

The random initial position of the grains leaves them poorly positioned in the mesh. There will be a great deal of overlapping and empty space. The simulated annealing algorithm repositions the grains so that the number of empty elements and overlapped elements is minimized. It operates by randomly perturbing the configuration of the grains and retaining the perturbation only if the configuration is improved or with a probability based on the “temperature”, which decreases at each time step.

randomly. The position of the grain can be changed by one element in any of six directions. A rotation is  $\pm 90^\circ$  around the x, y, or z-axis, and is achieved by multiplying each individual element coordinate by a rotation matrix. Since the element coordinates are relative to the global position of the element, the end effect is rotation of the entire grain.

An individual perturbation is kept if the configuration of the grains is improved. The change in the grain configuration is judged based on an evaluation function that counts the number of overlapped grains and empty elements, with empty elements weighted more heavily. If the value of this evaluation function is decreased, the perturbation is kept. If the value is unchanged or increased, then the perturbation is kept with probability

$$P = 1 - e^{-0.3T} \quad (3)$$

where T, the temperature, decreases with time from 1 to  $5 \times 10^{-3}$ .

After the grains are in their final configuration, orientations are assigned to the element mesh. There will still be many overlapping elements at this point that are unavoidable. Any grains which are redundant (entirely overlapped by other grains) are removed. Any remaining overlapped elements get the orientation of the last grain to claim them as the program reads through the grains sequentially. There will also be a small number of empty elements remaining in the final configuration. These elements are given the orientation of a randomly selected neighbor. In the rare case that an empty element is surrounded by other empty elements (this would only happen if grain size was very large), then the element is given an orientation selected from a random position in the mesh.

**Important Note:** By default, a coordinate rotation ( $90^\circ$  about y-axis and  $-90^\circ$  about x-axis) is applied to all the element orientations. To remove this rotation, comment out line 270.

## Output

Three files are written if the program is executed with an OIM scan as input. The first is the grain data file (extension .gex) discussed above. It is of the format in Figure 1, but only the Euler angles and radius are given non-zero values. The second file is a text file (extension .g) containing pictograms of the grain microstructure produced in the element mesh (see Figure 5). These pictograms are orthogonal slices of the microstructure along both the y-axis and z-axis. They are useful for quickly verifying that the microstructure generated has the desired grain sizes and shapes.

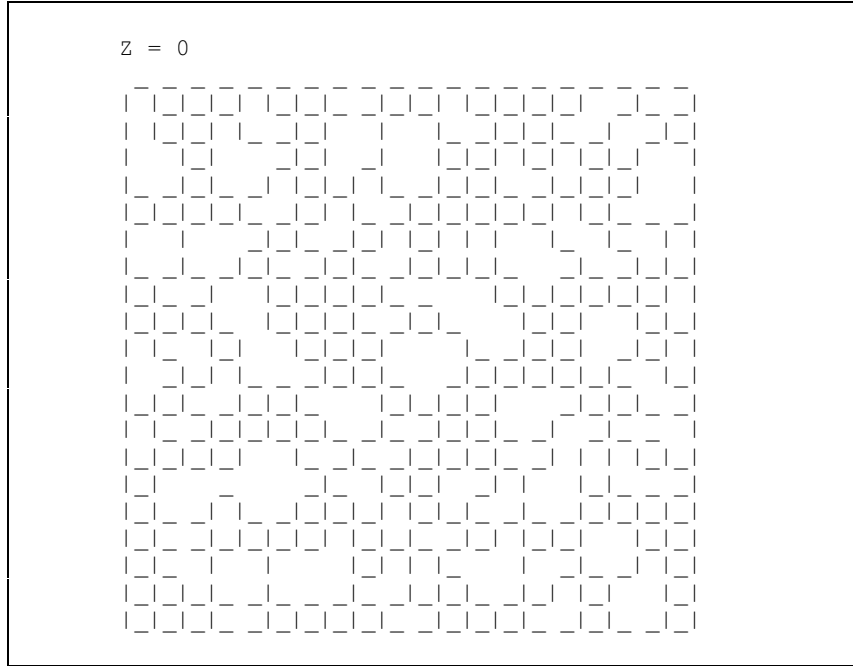


Figure 5. Example of grain pictogram of generated mesh.

The third file is the list of element orientations read by ALE3D. Its format is shown in Figure 6. Within Ale3D, MS\_CrystalOrientMethod 4 assigns rotations to each element according to its coordinates, matching the location of the element within the mesh to the corresponding location in the list of rotations in the OIMA3D output file. Therefore, it is essential that the coordinates and geometry of the mesh match exactly with the coordinates and geometry assumed by OIMA3D. If the mesh is a cube, all sides should be unit length. Otherwise, the smallest dimension length should be one, with the remaining lengths scaled accordingly.

X length	Y length	Z length	0	0
Phi1	PHI	phi2	0	0
.	.	.	0	0
.	.	.	0	0
# of Elem in X	# of Elem in Y	# of Elem in Z	0 (unused)	0 (unused)

Figure 6. Format of orientation output file (input for Ale3d).

## **MAKEJOBS – a script to MAKE six ALE3D JOBS**

### **Overview**

MAKEJOBS is a shell script which creates the six sets of input files, batch scripts, and restart files needed for the six ALE3D simulations used to compute K.

### **Usage**

MAKEJOBS <problem name> <orientation file> <number of processors>

### **Input**

Use of MAKEJOBS requires two input files: ‘makejobs.in’ and ‘makejobs.batch’. These files are the base versions of the input and batch files created by the script. They contain placeholders (&1, &2, ...) where relevant information such as paths or boundary conditions is inserted by the MAKEJOBS script. Do not remove these placeholders, but other modifications to the files can be made if different parameters are desired. A SAMI mesh file must also be present in the working directory with filename ‘<problem name>.sami’. This file is used when the script calls GENC. When the jobs run, the orientation file must also be present in the working directory.

### **Operation**

Figure 7 shows all the files needed to run MAKEJOBS and the directory structure and files created by the script.

```
/working directory
  makejobs
  makejobs.in
  makejobs.batch
  <prob>.sami
  <orientation file>
  svd_k
  <prob>xx.batch
  <prob>xy.batch
  <prob>xz.batch
  <prob>yy.batch
  <prob>yz.batch
  <prob>zz.batch
    /<prob>xx
      <prob>xx.in
      <prob>xx_00<np>_00000
    /<prob>xy
      <prob>xy.in
      <prob>xy_00<np>_00000
    /<prob>xz
      <prob>xz.in
      <prob>xz_00<np>_00000
    /<prob>yy
      <prob>yy.in
      <prob>yy_00<np>_00000
    /<prob>yz
      <prob>yz.in
      <prob>yz_00<np>_00000
    /<prob>zz
      <prob>zz.in
      <prob>zz_00<np>_00000
```

## SVD\_K – using Singular Value Decomposition to calculate K

### Overview

SVD\_K is a Matlab script that collects time history data from six ALE3D simulations and uses a singular value decomposition to compute K, the coefficient tensor used in ALE3D's anisotropic plasticity model.

### Usage

Within Matlab SVD\_K can be called directly, or invoking 'var = svd\_k' will store the calculated K to the Matlab environment variable 'var' for later use.

### Input

The required input for SVD\_K is six separate time history directories each containing the seven files:

e-mean   sx-mean   sy-mean   tyz-mean   txz-mean   txy-mean   mises-mean

Three user inputs are also required by the script: paths to the six time history directories, the deformation rate vector, and the plastic work value at which to perform the calculation. For the time history paths, there is an option to either type in the six paths or if MAKEJOBS was used, the paths can be found automatically given the problem name and number of domains (provided SVD\_K is running in the working directory from MAKEJOBS' run). There is also an option to use the default deformation rate, rather than typing it in.

### Operation

Operation begins with reading the required data from the time history files. The desired e value is converted to a time, and all necessary data is collected from the files. Values are interpolated and derivatives are calculated as the slope of a linear regression to four points surrounding the desired time value.

Next the elastic component of the deformation rate is removed according to the formulation

$$\underline{\underline{\mathcal{D}}} = \underline{\underline{L}} : (\underline{\underline{d}} - \underline{\underline{d}}^p) \quad (4)$$

$$\underline{\underline{d}}^p = \underline{\underline{d}} - \underline{\underline{L}}^{-1} : \underline{\underline{\mathcal{D}}} \quad (5)$$

$$\underline{d}^p = \frac{3}{2} \frac{\underline{\sigma}}{\underline{\sigma}^2} K : \underline{\sigma} \quad (6)$$

The deformation rate was determined in equation 5, and  $\underline{\sigma}$  is a 36x21 matrix formed by concatenating six 6x21 matrices formed from each set of time history data. The six matrices take the form

$$[\sigma] = \begin{bmatrix} \bar{\sigma}_{11} & \bar{\sigma}_{22} & \bar{\sigma}_{33} & 2\bar{\sigma}_{23} & 2\bar{\sigma}_{13} & 2\bar{\sigma}_{12} \\ & \bar{\sigma}_{11} & & & & \\ & & \bar{\sigma}_{11} & & & \\ & & & \bar{\sigma}_{11} & & \\ & & & & \bar{\sigma}_{11} & \\ & & & & & \bar{\sigma}_{11} \end{bmatrix} \quad (7)$$

Equation 6 is then solved for  $\mathbf{K}$  by finding the pseudo-inverse of  $\underline{\boldsymbol{\sigma}}$  through singular value decomposition, resulting in the final solution:

$$\begin{aligned}\{K\}_{21 \times 1} &= [\sigma]_{21 \times 36}^{-1} \{\hat{d}^p\}_{36 \times 1} \\ &= [V]_{21 \times 21} [S]_{21 \times 21}^{-1} [U]_{21 \times 36}^T \{\hat{d}^p\}_{36 \times 1}\end{aligned}\quad (8)$$

where  $V$  and  $U$  are orthogonal and  $S$  is diagonal.

## Output

Three files are written during execution of SVD\_K: 'dp.mat', 'k.mat', and 'kinput.mat'. The file 'dp.mat' contains the deformation rate vector with the elastic part removed, 'k.mat' contains the matrix K, and 'kinput.mat' contains K in the form needed for an ALE3D input file. The input form of k is also printed to the Matlab display, so it can be directly pasted into the ALE3D input file.